Perbandingan Algoritma A*, UCS, dan Greedy Best First Search pada Pencarian Solusi Permainan Words of Wonders menggunakan Regex

Aurelius Justin Philo Fanjaya - 13522020
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522020@std.stei.itb.ac.id

***Abstract*—Algoritma A*, UCS (Uniform Cost Search), dan Greedy Best First Search dapat diimplementasikan untuk menlakukan *pathfinding* atau mencari jalur dari berbagai macam persoalan. Salah satu persoalan yang dapat diselesaikan oleh ketiga algoritma ini adalah untuk mencari solusi permainan Words of Wonders. Pencarian solusi dari permasalahan ini juga akan menggunakan Regex (Regular Expression) untuk menentukan kelayakan kata untuk digunakan pada sel yang tersisa. Makalah ini akan membahas perbandingan antara algoritma A*, UCS, dan Greedy Best First Search dalam pencarian solusi permainan Words of Wonders menggunakan Regex.**

Keywords—algoritma, A*, UCS, Greedy Best First Search, Regex

I. PENDAHULUAN

Permainan Words of Wonders adalah salah satu permainan *puzzle* yang berkaitan dengan huruf dan kata. Dalam permainan ini, pemain ditugaskan untuk mengisi petak-petak yang kosong (seperti teka-teki silang/*crossword*), hanya dengan menggunakan huruf-huruf yang disediakan pada suatu level. Berikut adalah contoh tampilan permainan Words of Wonders.

Gambar 1.1. Contoh tampilan permainan Words of Wonders (diambil dari *game* Words of Wonders langsung)

Berdasarkan gambar tersebut, misalkan pemain diberikan huruf-huruf P, L, A, dan N untuk digunakan dalam mengisi kata-kata pada crossword yang diberikan, maka pemain hanya dapat menggunakan huruf-huruf tersebut masing-masing sekali. Selain itu, 1 kata hanya dapat digunakan sekali untuk mengisi *crossword*. Pada gambar tersebut, beberapa dari kata yang menjadi solusi dari *crossword* tersebut adalah PLAN, LAP, dan NAP yang merupakan kombinasi dari huruf-huruf P, L, A, dan N.

Algoritma UCS, A*, dan Greedy Best First Search dapat digunakan dalam penyelesaian persoalan ini karena ketiga algoritma tersebut dapat melakukan *pathfinding* atau pencarian jalur, yang pada kasus ini adalah pemilihan kata-kata yang terdiri dari huruf-huruf tertentu dan dapat menjadi solusi dari *crossword* tersebut.

II. LANDASAN TEORI

A. *Penentuan Rute (Route/Path Planning)*

Penentuan Rute (*Route/Path Planning*) adalah salah satu permasalahan dalam traversal graf. Pencarian dengan melakukan traversal graf dapat dilakukan dengan berbagai macam cara. Secara umum, algoritma pencarian yang digunakan dalam pencarian rute dibagi menjadi 2 kelompok, yaitu Uninformed Search dan Informed Search, dengan penjelasan sebagai berikut.

1. Uninformed Search
   Uninformed Search atau *Blind Search* adalah algoritma pencarian tanpa adanya informasi tambahan mengenai *goal* atau tujuan dari pencarian. Contoh dari algoritma *Uninformed Search* adalah sebagai berikut.
   - Breadth First Search (BFS)
   - Depth First Search (DFS)
   - Depth Limited Search (DLS)
   - Iterative Deepening Search (IDS)
   - Uniform Cost Search (UCS)

2. Informed Search

Informed Search adalah algoritma pencarian dengan adanya informasi tambahan mengenai *goal* atau tujuan dari pencarian. Informasi tambahan ini dapat berupa jarak simpul tertentu pada graf ke simpul tujuan pencarian, ataupun informasi lainnya mengenai tujuan pencarian yang dapat membantu proses pencarian. Contoh dari algoritma *Informed Search* adalah sebagai berikut.

- Greedy Best First Search (GBFS)
- A* (A-star)

B. Algoritma Uniform Cost Search

Algortma *Uniform Cost Search* atau UCS adalah salah satu algoritma *route planning* atau penentuan rute tanpa adanya informasi tambahan mengenai tujuan pencarian (*Uninformed Search*/*Blind Search*). Pencarian dilakukan berdasarkan *cost* atau ongkos untuk sampai ke suatu simpul pada graf. *Cost* atau ongkos pada UCS biasanya dinotasikan sebagai *g(n)*, di mana n adalah simpul pada graf pencarian UCS. Pada pencarian *cost* minimum, simpul yang yang memiliki *cost* paling rendah hingga sampai ke suatu titik akan dibangkitkan terlebih dahulu.

Pada proses penentuan *cost*, tidak diketahui informasi tambahan apapun mengenai tujuan pencarian, sehingga algoritma ini masuk ke dalam *Uninformed Search*. Pada algoritma ini, simpul yang memiliki *cost* paling rendah akan dibangkitkan terlebih dulu, maka implementasi dari algoritma ini biasanya menggunakan *Priority Queue*, di mana simpul paling yang memiliki prioritas paling tinggi akan berada di paling depan.

Nilai prioritas dari algoritma ini sama dengan ongkos, sehingga digunakan persamaan sebagai berikut.

$$f(n) = g(n),$$

di mana *f(n)* adalah fungsi evaluasi yang menjadi nilai prioritas suatu simpul dan *g(n)* adalah nilai *cost* atau ongkos suatu simpul.

C. Algoritma Greedy Best First Search

Algoritma Greedy Best First Search adalah salah satu algoritma *route planning* atau penentuan rute dengan adanya informasi tambahan mengenai tujuan pencarian (Informed Search). Pencarian dilakukan berdasarkan nilai heuristik (*h(n)*) suatu simpul. Nilai heuristik adalah estimasi ongkos dari simpul n ke simpul tujuan. Misalnya, nilai heuristik dalam penentuan jarak suatu tempat ke tempat lainnya dapat diestimasi sebagai jarak garis lurus dari kedua tempat tersebut. Pada algoritma Greedy Best First Search, simpul dengan nilai heuristik terbaik (terkecil dalam kasus pencarian ongkos minimum) akan dibangkitkan terlebih dahulu.

Pada algoritma ini, simpul yang memiliki nilai heuristik paling rendah akan dibangkitkan terlebih dulu, maka implementasi dari algoritma ini biasanya menggunakan *Priority Queue*, di mana simpul paling yang memiliki prioritas paling tinggi akan berada di paling depan.

Nilai prioritas dari algoritma ini sama dengan nilai heuristik suatu simpul, sehingga digunakan persamaan sebagai berikut.

$$f(n) = h(n),$$

di mana *f(n)* adalah fungsi evaluasi yang menjadi nilai prioritas suatu simpul dan *h(n)* adalah nilai heuristik yang merupakan estimasi ongkos jarak dari simpul n menuju simpul tujuan.

*D. Algoritma A**

Algoritma A* (A-star) adalah salah satu algoritma *route planning* atau penentuan rute dengan adanya informasi tambahan mengenai tujuan pencarian (Informed Search). Algoritma A* adalah pengembangan dari algoritma *Uniform Cost Search* dan *Greedy Best First Search*, dengan ide untuk menghindari *path* (jalur) atau *simpul* yang sudah mahal selain menggunakan nilai heuristik. Pencarian dilakukan berdasarkan fungsi evaluasi (*f(n)*) suatu simpul. Fungsi evaluasi ini adalah penjumlahan dari ongkos untuk sampai ke suatu simpul ditambahkan dengan nilai heuristik simpul tersebut untuk sampai ke simpul tujuan. Pada algoritma A*, simpul dengan nilai fungsi evaluasi terbaik (terkecil dalam kasus pencarian ongkos minimum) akan dibangkitkan terlebih dahulu.

Pada algoritma ini, simpul yang memiliki nilai fungsi evaluasi paling rendah akan dibangkitkan terlebih dulu, maka implementasi dari algoritma ini biasanya menggunakan *Priority Queue*, di mana simpul paling yang memiliki prioritas paling tinggi akan berada di paling depan.

Nilai prioritas dari algoritma ini sama dengan nilai heuristik suatu simpul, sehingga digunakan persamaan sebagai berikut.

$$f(n) = g(n) + h(n),$$

di mana *f(n)* adalah fungsi evaluasi yang menjadi nilai prioritas suatu simpul, *g(n)* adalah nilai *cost* atau ongkos suatu simpul, dan *h(n)* adalah nilai heuristik yang merupakan estimasi ongkos jarak dari simpul n menuju simpul tujuan.

Pada Algoritma A*, terdapat konsep *admissible heuristics*, yaitu fungsi heuristik (*h(n)*) yang selalu lebih kecil dari dari ongkos sebenarnya dari simpul n menuju simpul tujuan. Jika suatu fungsi heuristik yang dipakai dalam pencarian *admissible*, maka A* dijamin menghasilkan solusi yang optimal.

E. Regex (Regular Expression)

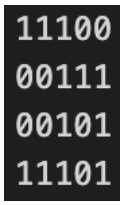
Regex atau Regular Expression salah satu jenis *pattern/string matching*. *String* matching adalah algortima atau proses pencocokkan suatu *pattern* terhadap suatu text, dan mencari keberadaan *pattern* tersebut dalam text. Pada *exact matching*, dicari bagian dari text yang sama persis dengan *pattern*. Sedangkan pada *regex matching*, dicari bagian dari text yang menyerupai atau cocok dengan suatu *pattern regex* tertentu. Suatu *pattern regex* dapat cocok dengan berbagai substring berbeda yang terdapat pada suatu text.

III. IMPLEMENTASI ALGORITMA A*, UCS, DAN GREEDY BEST FIRST SEARCH DALAM PERMAINAN WORDS OF WONDERS

A. Representasi Permasalahan dalam Program

1. *Crossword* atau Teka-Teki Silang

   Crossword atau teka-teki silang pada permainan direpresentasikan sebagai array dua dimensi karakter atau matriks karakter. Sebagai input program, matriks akan direpresentasikan oleh matriks yang berisikan angka 1 atau 0, di mana angka 1 merepresentasikan sel kosong yang dapat diisi sedangkan angka 0 adalah bagian yang tidak dapat diisi. Gambar berikut adalah representasi matrix dari level permainan pada *Gambar 1.1*.



Gambar 3.1. Representasi matrix crossword Gambar 1.1

   Matrix tersebut akan diproses sehingga angka 1 pada matrix tersebut berubah menjadi karakter '.'. Hal ini dilakukan karena pada pattern regex pada python, karakter '.' Merepresentasikan semua kemungkinan karakter sehingga menandakan sel tersebut belum terisi. Setelah diproses, tiap kolom yang direpresentasikan angka 1 dapat diisi oleh sebuah karakter jika sudah terdapat kata yang mengisi sel tersebut secara horizontal maupun vertikal. Berikut adalah contoh matriks dari Gambar 3.1 yang sudah diproses dan diisi oleh beberapa kata.

Gambar 3.2. Matrix crossword yang sudah terisi

   Pada matrix tersebut, huruf terisi sesuai dengan tempatnya pada matriks, angka 0 direpresentasikan whitespace, dan karakter '.' direpresentasikan blok berwarna putih.

2. Simpul Graf

   Pada program, simpul dalam pohon pencarian merepresentasikan suatu struktur data *State*. State merepresentasikan kata mana saja yang sudah terisi pada simpul tersebut. Dalam Struktur data *State* ini, disimpan informasi berupa matrix pada simpul tersebut, *f(n)* atau fungsi evaluasi dari simpul tersebut, parent dari simpul tersebut, daftar kata pada simpul tersebut, index dari kata yang terakhir diisi, dan set kata-kata apa saja yang sudah pernah digunakan. Pada awal pencarian, simpul pertama memiliki matriks dengan setiap sel yang dapat diisi dalam masih kosong, ongkos 0, daftar kata kosong, dan index kata yang terkahir diisi -1 (belum ada yang diisi).

3. Daftar kata atau kamus yang digunakan

   Dalam implementasi program, kata-kata yang dapat digunakan diambil dari kamus pada tautan berikut.

   https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt

B. Implementasi Program

Proses pencarian solusi yang diimplementasikan pada program ini dalam bahasa python secara umum adalah sebagai berikut.

1. Program memasukkan daftar kata yang dapat digunakan dari dictionary ke dalam sebuah set.
2. Program meminta input user berupa nama file yang akan menjadi input *crossword*, huruf apa saja yang dapat digunakan, dan algoritma yang ingin digunakan.
3. Program membuat semua kemungkinan kata yang dapat digunakan dari huruf-huruf yang dapat digunakan dan berada di dalam kamus.
4. Program menginstansiasi State awal, dengan parameter:
   - matrix: matriks input yang masih belum terisi
   - parent: None
   - wordList: daftar kata yang masih terdiri atas kumpulan karakter '.' Saja.
   - index: -1 (belum pernah mengisi apa-apa)
   - visited: set kosong
5. Program melakukan pencarian sesuai dengan algoritma yang digunakan (UCS/GBFS/A*), dengan State awal yang sudah didefinisikan pada poin 4.
6. Algoritma pencarian mengembalikan *path* solusi berupa State pada tiap simpul yang menjadi solusi, dan menampilkan (print) matrix pada tiap State tersebut.
7. Program menampilkan waktu eksekusi dan total jumlah simpul yang dibangkitkan selama pencarian.

Penjelasan dari poin no 5 proses pencarian solusi, yaitu implementasi masing-masing algoritma (Uniform Cost Search, Greedy Best First Search, dan A*) cukup mirip. Secara umum, ketiga algoritma tersebut akan melakukan proses sebagai berikut.

1. Jika priority queue tidak kosong, algoritma melakukan pop terhadap priority queue

(menggunakan heapq) yang terurut berdasarkan priority (evaluation function).
2. Jika priority queue kosong, return None.
3. Algoritma menambah/*increment* jumlah *node visited* sebanyak 1.
4. Algoritma mengecek apakah simpul sudah menjadi solusi, yaitu jika index sudah sama dengan index terakhir pada wordList.
5. Jika simpul merupakan solusi, return path dari start State (simpul awal) ke simpul tersebut.
6. Jika simpul bukan merupakan solusi, tambahkan kata pada simpul kepada kata yang sudah pernah dikunjungi, kemudian bangkitkan possible next move dari simpul tersebut.
7. Kembali ke tahap 1.

Perbedaan dari ketiga algoritma adalah perhitungan nilai priority yang digunakan untuk mengurutkan priority queue pada algoritma. Nilai priority queue yang digunakan adalah nilai fungsi evaluasi (*f(n)*) pada suatu simpul, dengan penjelasan sebagai berikut.

1. Uniform Cost Search

   $$f(n) = g(n),$$

   di mana *g(n)* adalah jumlah simpul yang sudah dilewati dari simpul awal ke simpul n.

2. Greedy Best First Search

   $$f(n) = h(n),$$

   di mana *h(n)* adalah heuristik yang dihitung dari jumlah simpul yang harus dilewati dari simpul n untuk sampai ke simpul tujuan.

3. A*

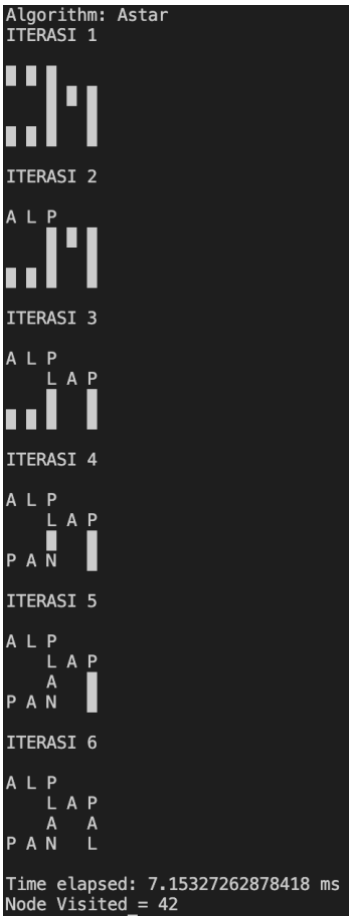
   $$f(n) = g(n) + h(n),$$

   di mana *g(n) dan h(n)* sama seperti kedua algoritma yang dijelaskan sebelumnya, yaitu *g(n)* adalah jumlah simpul yang sudah dilewati dari simpul awal ke simpul n dan *h(n)* adalah heuristik yang dihitung dari jumlah simpul yang harus dilewati dari simpul n untuk sampai ke simpul tujuan.

Proses pembangkitan next word (simpul selanjutnya) pada program ini menggunakan Regex (Regular Expression) dengan cara sebagai berikut.

1. Menentukan semua kemungkinan anagram dari huruf-huruf input (misalkan anagram PLAN adalah PALN, NLAP, ALPN, LAPN, dan lainnya) menggunakan itertools.
2. Menetukan semua substring dari tiap anagram tersebut, dan memasukkan semuanya ke dalam sebuah set. (misalkan PLAN menjadi P, L, AN, PAN, PLN, dan lainnya).
3. Kata yang dapat digunakan adalah irisan dari set kata-kata pada kamus bahasa inggris (set Dictionary) dengan set substring anagram yang baru ditentukan tadi.
4. Pada suatu simpul, kata dari simpul anak yang dapat dibangkitkan dari simpul tersebut adalah kata yang sesuai dengan regex pada simpul tersebut. Misalnya, pada simpul tersebut sudah terisi “S . . H .“, maka huruf pertama dari huruf yang dibangkitkan harus diawali huruf S, dan huruf keempatnya adalah H, contohnya “SIGHT”.
5. Pattern Regex yang digunakan dalam contoh pada poin nomor 4 adalah ” \AS..H.\Z”, di mana \A berarti string harus diawali dengan pattern tersebut, dan \Z berarti string harus diakhiri oleh pattern tersebut.
6. Semua kata pada irisan set pada poin nomor 3 yang match dengan pattern regex pada poin nomor 5 adalah kata-kata yang dibangkitkan untuk next word.

Contoh keluaran berupa path (matriks pada tiap iterasi) hasil dari algoritma pada program yang digunakan beserta waktu eksekusi dan jumlah *node visited*-nya adalah sebagai berikut.



*Gambar 3.3. Contoh hasil pencarian Algoritma A**

IV. PENGUJIAN

Berdasarkan program yang telah dibuat, dilakukan pengujian dengan kasus-kasus dan hasil sebagai berikut.

1. Kasus 1

Crossword	1110111
Huruf-huruf yang dapat digunakan	TCA

Tabel 4.1.1 Tabel Input Kasus 1

Algoritma	UCS	GBFS	A*
Node Visited	4	3	4
Waktu Eksekusi (ms)	0.841856	0.391244	0.583887
Matriks Solusi	A C T C A T	A C T C A T	A C T C A T

Tabel 4.1.2 Tabel Hasil Kasus 1

2. Kasus 2

Crossword	1111111 0000001 0000111 0010100 0010111 1111001 0101001 0111001
Huruf-huruf yang dapat digunakan	BAUDRZZ

Tabel 4.2.1 Tabel Input Kasus 2

Algoritma	UCS	GBFS	A*
Node Visited	39604	9229	17594
Waktu Eksekusi (ms)	2086.898088	357.921123	692.283153
Matriks Solusi	B U Z Z A R D A R U B B A U D U B D A R B R R A A B U D D	B U Z Z A R D A U R B B R R D U B D R A B U U U R B A R A	B U Z Z A R D A U R B B R U D U B D A R B R R U A B A D D

Tabel 4.2.2 Tabel Hasil Kasus 2

3. Kasus 3

Crossword	111110 100100 100111 111001 010001 010001
Huruf-huruf yang dapat digunakan	THIGS

Tabel 4.3.1 Tabel Input Kasus 3

Algoritma	UCS	GBFS	A*
Node Visited	112	26	29
Waktu Eksekusi (ms)	21.446943	3.710985	4.413843
Matriks Solusi	S I G H T I I G S I T H I T H T I S S	S I G H T H I I S I T T I S H T I S S	S I G H T I I G S I T H I T H T I S S

Tabel 4.3.2 Tabel Hasil Kasus 3

4. Kasus 4

Crossword	11100 00111 00101 11101
Huruf-huruf yang dapat digunakan	PLAN

Tabel 4.4.1 Tabel Input Kasus 4

Algoritma	UCS	GBFS	A*
Node Visited	89	52	42
Waktu Eksekusi (ms)	9.913206	4.074096	3.644943
Matriks Solusi	A L P L A P A A P A N L	A L P L A P A A P A N L	A L P L A P A A P A N L

Tabel 4.4.2 Tabel Hasil Kasus 4

5. Kasus 5

Crossword	1110000 1010100 1011111 0000101 0001111
Huruf-huruf yang dapat digunakan	GULDE

Tabel 4.5.1 Tabel Input Kasus 5

Algoritma	UCS	GBFS	A*
Node Visited	1035	375	133
Waktu Eksekusi (ms)	53.211688	27.625083	12.055158
Matriks Solusi			

Tabel 4.5.2 Tabel Hasil Kasus 5

Tabel Rata-Rata

Algoritma	UCS	GBFS	A*
Node Visited	8168.8	1937	3560.4
Waktu Eksekusi (ms)	434.4623562	78.7445062	142.5961968

Tabel 4.1 Tabel Rata-Rata

V. ANALISIS

Berdasarkan Pengujian, didapatkan bahwa urutan algoritma dengan rata-rata waktu eksekusi serta jumlah *node visited* atau jumlah simpul yang dikunjungi terurut menaik adalah *Greedy Best First Search*, A*, dan UCS. Hal ini menandakan bahwa dalam permasalahan ini, yaitu pencarian solusi permainan Words of Wonders, Greedy Best First Search adalah algoritma yang terbaik untuk mayoritas kasus, disusul oleh Algoritma A* dan UCS.

Pada kasus 1, didapatkan bahwa urutan algoritma dengan jumlah *node visited* dan waktu eksekusi terurut secara menaik adalah GBFS, A*, dan UCS. Untuk kasus yang sederhana, dalam kasus 1 ini ditandai dengan huruf-huruf yang dapat digunakan sejumlah 3 saja dan *Crossword* yang hanya dapat diisi 2 huruf, maka sebenarnya jumlah node visited dan waktu eksekusi sangatlah mirip. UCS dan A* memiliki jumlah *node visited* yang sama yaitu 4 *node* dan GBFS hanya selisih 1 dengan 3 *node.*

Pada kasus 2, didapatkan bahwa urutan algoritma dengan jumlah *node visited* dan waktu eksekusi terurut secara menaik adalah GBFS, A*, dan UCS. Urutan tersebut sama dengan kasus 1, meskipun kasus 2 ini termasuk dalam kasus yang kompleks. Dalam kasus ini, terdapat 7 huruf yang dapat digunakan serta 11 kata yang harus diisi dalam *crossword*. Meskipun urutannya sama, namun dapat dilihat bahwa angka *node visited* serta waktu eksekusi ketiga algoritma tersebut cukup jauh. Algoritma A* membutuhkan jumlah *node visited* serta waktu eksekusi sekitar 2 kali lipat dari GBFS, dan algoritma UCS membutuhkan jumlah node visited sekitar 4 kali GBFS serta waktu eksekusi sekitar 6 kali lipat dari GBFS.

Pada kasus 3, didapatkan bahwa urutan algoritma dengan jumlah *node visited* dan waktu eksekusi terurut secara menaik adalah GBFS, A*, dan UCS. Urutan tersebut masih sama dengan kasus 1 dan 2. Kompleksitas untuk kasus 3 ini cukup sedang karena hanya memiliki 4 pilihan huruf yang dapat digunakan dan 7 kata pada *crossword* yang harus diisi. GBFS dan A* memiliki jumlah *node visited* dan waktu eksekusi yang hampir sama, walalupun GBFS masih sedikit lebih cepat dibandingkan A*. Algoritma GBFS membutuhkan jumlah *node visited* sekitar 4 kali lipat dari GBFS dan waktu eksekusi sekitar 6 kali lipat dari GBFS.

Urutan pada kasus 4 cukup menarik karena berbeda dari kasus 1, 2, dan 3. Urutan algoritma dengan jumlah *node visited* dan waktu eksekusi terurut secara menaik adalah A*, GBFS, dan UCS. Jumlah node visited dan waktu eksekusi A* lebih sedikit dibandingkan GBFS, dan angka dari algoritma UCS sebenarnya tidak terlalu jauh dari kedua algoritma tersebut (sekitar 2 kali lipat dari A* dan GBFS). Kompleksitas dari persoalan pada kasus ini dapat dibilang cukup mudah karena hanya terdapat 4 pilihan huruf yang dapat digunakan dan hanya 5 kata yang harus diisi pada *crossword.*

Urutan pada kasus 5 sama dengan kasus 4, yaitu A*, GBFS, dan UCS, hanya saja rasio atau perbandingan dari hasil ketiga algoritma tersebut yang semakin berbeda. Pada kasus 5 ini algoritma A* hanya melakukan 133 *node visited*, sedangkan GBFS 375 *node*, dan UCS 1035 *node*. Hal ini menunjukkan bahwa pada kasus ini, algoritma A* cukup jauh lebih baik dalam mencari solusi dibandingkan UCS maupun GBFS. Kompleksitas dari kasus ini pun dapat dibilang sedang, dengan 5 pilihan huruf yang dapat digunakan serta 7 kata yang harus diisi pada *crossword*.

Berdasarkan analisis kasus-kasus tersebut, dapat dilihat bahwa algoritma Greedy Best First Search secara cukup konsisten menjadi algoritma yang terbaik dalam mencari solusi dari permainan Words of Wonders ini. Hanya saja, terdapat kasus-kasus tertentu seperti pada kasus 4 dan 5 di mana A* dapat menghasilkan hasil yang lebih baik dibandingkan Greedy Best First Search. Hal ini disebabkan karena algoritma Greedy Best First Search akan selalu memilih 1 *path*/jalur terlebih dulu hingga terjadi *deadend* atau tidak dapat ditemukan solusi pada jalur tersebut. Jika terlalu banyak *deadend* terjadi dalam pencarian, maka algoritma GBFS akan membangkitkan lebih banyak simpul untuk menemukan solusi.

VI. KESIMPULAN

Berdasarkan analisis terhadap kasus-kasus yang telah dilakukan pengujian, didapatkan kesimpulan bahwa algoritma Greedy Best First Search adalah algoritma yang paling konsisten memberikan hasil yang terbaik dalam pencarian solusi permainan, disusul oleh algoritma A* dan terakhir UCS. Algoritma A* memberikan hasil yang lebih baik dibandingkan algortima Greedy Best First Search pada kasus tertentu, sedangkan algoritma Uniform Cost Search (UCS) secara konsisten memberikan hasil yang lebih buruk dibandingkan kedua algoritma lainnya, terutama pada kasus yang kompleksitasnya sedang atau kompleks.

SOURCE CODE REPOSITORY LINK AT GITHUB

https://github.com/AureliusJustin/TugasMakalahStima_13522020.git

UCAPAN TERIMA KASIH

Terima kasih saya panjatkan kepada Tuhan yang Maha Esa karena berkat-Nya penulis dapat menyelesaikan makalah Strategi Algoritma ini tanpa kendala yang berarti. Tak lupa penulis juga berterimakasih kepada Dr. Ulfa Malidevi, S.T., M.Sc. selaku dosen Strategi Algoritma untuk K02 atas bimbingannya selama 1 Semester terakhir dalam mata kuliah IF2211 Strategi Algoritma. Penulis berharap makalah ini dapat bermanfaat dan menjadi referensi di masa depan.

REFERENSI

[1] Fugo Games, Words of Wonders: Crossword. (accessed 12 June 2024)

[2] Maulidevi, Nur Ulfa. "Penetuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search, Bahan Kuliah IF2211 Strategi Algoritma," 2021. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf (accessed 12 June 2024)

[3] Maulidevi, Nur Ulfa. "Penetuan Rute (Route/Path Planning) Bagian 2: Algoritma A*, Bahan Kuliah IF2211 Strategi Algoritma," 2021. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf (accessed 12 June 2024)

[4] Khodra, Masayu Leylia. "String Matching dengan Regular Expression," Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf (accessed 12 June 2024)

[5] Oracle. https://docs.oracle.com/javase/tutorial/collections/interfaces/examples/dictionary.txt (accessed 12 June 2024)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Aurelius Justin Philo Fanjaya 13522020